MPI Functions

MPI_Send and MPI_Recv are the basic blocking send and receive commands in MPI.

MPI_Send has the following six parameters:

1. The **data that you wish to send**. note that you must have the address of the variable and not the value itself.
2. The **count of items to send**. if you have more than one item to send at a time you specify the count here.
3. The **data type** that is being communicated over the network. There are default basic data types but it is possible to define your own data types.
4. The **rank of the process to receive** the message.
5. The **message tag**. This gives you a method of distinguishing between different message types.
6. The **communication group** that the message is to be sent to. Because our programs will be small we will use MPI_COMM_WORLD a lot.

```
MPI_Send(&dataToSend, count, dataType, rankSendingTo, tag, commGroup)
```

```
MPI_Recv(&dataRecv, count, dataType, rankOrigin, tag, commGroup, Status)
```

MPI_Recv has the same first six parameters as MPI_Send but with minor differences.

The **fourth parameter** is now the process that you are **receiving from** not sending to.

The **last parameter** indicates the status of the message that has been received. For something as simple as this we will ignore the status information for now.

MPI_Bcast takes 5 arguments:

1. The **buffer** to send from or recieve into
2. The **count** of items to send.
3. The **type of data** to send.
4. The **root of this communication**. If the root matches the world rank MPI_Bcast will initiate a send. Otherwise it will initiate a receive.
5. The **communicator** through which this message will be sent.

```
MPI_Bcast(&dataSend/Recv, count, dataType, root, commGroup)
```

MPI_Reduce requires 7 arguments:

1. The **local variable** that is to be reduced across all nodes. All nodes must implement this variable.
2. The **destination variable** that will contain the reduced value. All nodes must implement this variable but only the root node will store a value in this variable.
3. The **count**
4. The **type**
5. The **reduce operation**, to be performed on all nodes. Sum is one such built in operation but there are other built in operations like min, max, bitwise and/or/not, logical and/or/not etc.
6. The sixth argument is the **root of the operation**. This functions in the same manner as MPI_Bcast.
7. The **communicator** on which the operation is performed

```
MPI_Reduce(&localVar, &destVar, count, dataType, operation,
root, commGroup)
```

MPI_Barrier forces synchronisation. It accepts a single parameter which is a communicator. The barrier will force all nodes into a cold sleep until all nodes in the communicator have made the MPI_Barrier call. MPI_Barrier will then release all nodes to start computing again. Hence this is why you see the synchronised message at the end of computation.

The **MPI_Scatter** command expects 8 arguments

1. The **array that is to be scattered** amongst the ranks in the communicator. ***

2. The **number of items** that should be distributed to each rank ***

3. The **type of data** that is being distributed ***

4, 5, 6: follow the same format as the first three but this time they describe the buffer that is receiving the data. again a location, count and data type that is being received ***

7, 8: Specify the **root of the scatter** & **communicator** that this scatter is functioning on.

*** Referenced by root node
*** Referenced by non-root

```
MPI_Scatter(&totalArray*, count*, dataType*, &partialArray*, count*, dataType*, root, commGroup)
```

MPI_Gather: takes the same number of arguments and the same types for each argument as MPI_Scatter

be careful though the size to send must match the size to receive as the root node is expecting this size from all nodes that are taking part in the gather. If you specify the total size of the partition in the recieve your MPI application will crash and will be shut down.

```
MPI_Gather(&partialArray, count, dataType, &totalArray, count, dataType, root, commGroup)
```